# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №2

По дисциплине: «**Конструирование программ и языки программирования**»

Группа ПО-455

## Выполнил О. Е. Листопадова

Шифр 21

# 2022

**22 Охарактеризуйте класс WebClient**

Если необходимо только запросить файл с определенного URI (Uniform Resource Identifier — унифицированный идентификатор ресурса), то простейшим в использовании классом .NET, который подходит для этого, будет **System.Net.WebClient**. Этот исключительно высокоуровневый класс предназначен для выполнения базовых операций с помощью всего одной или двух команд. В настоящее время в .NET Framework поддерживаются URI, начинающиеся с идентификаторов *http:*, *https:* и *file:*.

Важно отметить, что термин URL (Uniform Resource Locator — универсальный локатор ресурсов) больше не используется в новых технических спецификациях, а вместо него отдается предпочтение URI. URI имеет приблизительно тот же смысл, что и URL, но немного более общий, потому что в URL не подразумевается обязательное применение одного из знакомых протоколов, таких как HTTP или FTP.

**Свойства, определенные в классе WebClient**

**public string BaseAddress { get; set; } -** Получает или устанавливает базовый адрес требуемого URI. Если это свойство установлено, то адреса, задаваемые в методах класса **WebClient,** должны определяться относительно этого базового адреса

**public RequestCachePolicy CachePolicy { get; set; } -** Получает или устанавливает правила, определяющие, когда именно используется кэш

**public ICredentials Credentials { get; set; } -** Получает или устанавливает мандат, т.е. учетные данные пользователя. По умолчанию это свойство имеет пустое значение

**public Encoding Encoding { get; set; } -** Получает или устанавливает схему кодирования символов при передаче строк

**public WebHeaderCollection Headers{ get; set; } -** Получает или устанавливает коллекцию заголовков запроса

**public bool IsBusy { get; } -** Принимает логическое значение **true,** если данные по-прежнему передаются по запросу, а иначе — логическое значение **false**

**public IWebProxy Proxy { get; set; } -** Получает или устанавливает прокси-сервер

**public NameValueCollection QueryString { get; set; } -** Получает или устанавливает строку запроса, состоящую из пар “имя-значение”, которые могут быть присоединены к запросу. Строка запроса отделяется от URI символом ?. Если же таких пар несколько, то каждая из них отделяется символом **@**

**public WebHeaderCollection ResponseHeaders{ get; } -** Получает коллекцию заголовков ответа

**public bool UseDefaultCredentials { get; set; } -** Получает или устанавливает значение, которое определяет, используется ли для аутентификации устанавливаемый по умолчанию мандат. Если принимает логическое значение **true,** то используется мандат, устанавливаемый по умолчанию, т.е. учетные данные пользователя, в противном случае этот мандат не используется

**Методы синхронной передачи, определенные в классе WebClient**

**public byte[] DownloadData(string** *address) -* Загружает информацию по адресу URI, обозначаемому параметром *address.* Возвращает результат в виде массива байтов

**public byte[] DownloadData(Uri** *address) -* Загружает информацию по адресу URI, обозначаемому параметром *address.* Возвращает результат в виде массива байтов

**public void DownloadFile(string** *uri,* **string** *fileName) -* Загружает информацию по адресу URI, обозначаемому параметром *fileName.* Сохраняет результат в файле *fileName*

**public void DownloadFile(Uri** *address,***string** *fileName) -* Загружает информацию по адресу URI, обозначаемому параметром *address.* Сохраняет результат в файле *fileName*

**public string DownloadString(string** *address) -* Загружает информацию по адресу URI, обозначаемому параметром *address.* Возвращает результат в виде символьной строки типа **string**

**public string DownloadString(Uri** *address) -* Загружает информацию по адресу URI, обозначаемому параметром *address.* Возвращает результат в виде символьной строки типа **string**

**public Stream OpenRead(string** *address) -* Возвращает поток ввода для чтения информации по адресу URI, обозначаемому параметром *address.* По окончании чтения информации этот поток необходимо закрыть

**public Stream OpenRead(Uri** *address) -* Возвращает поток ввода для чтения информации по адресу URI, обозначаемому параметром *address.* По окончании чтения информации этот поток необходимо закрыть

**public Stream OpenWrite(string** *address) -* Возвращает поток вывода для записи информации по адресу URI, обозначаемому параметром *address.* По окончании записи информации этот поток необходимо закрыть

**public Stream OpenWrite(Uri** *address) -* Возвращает поток вывода для записи информации по адресу URI, обозначаемому параметром *address.* По окончании записи информации этот поток необходимо закрыть

**public Stream OpenWrite(string** *address,* **string** *method) -* Возвращает поток вывода для записи информации по адресу URI, обозначаемому параметром *address.* По окончании записи информации этот поток необходимо закрыть. В строке, передаваемой в качестве параметра *method*, указывается, как именно следует записывать информацию

**public Stream OpenWrite(Uri** *address,***string** *method) -* Возвращает поток вывода для записи информации по адресу URI, обозначаемому параметром *address.* По окончании записи информации этот поток необходимо закрыть. В строке, передаваемой в качестве параметра *method*, указывается, как именно следует записывать информацию

**public byte[] UploadData(string** *address,* **byte[]** *data) -* Записывает информацию из массива *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public byte[] UploadData(Uri** *address,***byte[]** *data) -* Записывает информацию из массива *data* по адресу URI, 'обозначаемому параметром *address.* В итоге возвращается ответ

**public byte[] UploadData(string** *address,* **string** *method,***byte[]** *data) -* Записывает информацию из массива *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method,* указывается, как именно следует записывать информацию

**public byte[] UploadData(Uri** *address,***string** *method,* **byte[]** *data) -* Записывает информацию из массива *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method,* указывается, как именно следует записывать информацию

**public byte[] UploadFile(string** *address,* **string** *fileName) -* Записывает информацию в файл *fileName* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public byte[] UploadFile(Uri** *address,***string** *fileName) -* Записывает информацию в файл *fileName* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public byte[] UploadFile (string** *address,* **string** *method,***string** *fileName) -* Записывает информацию в файл *fileName* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method***,** указывается, как именно следует записывать информацию

**public byte[] UploadFile(Uri** *address,***string** *method***, string** *fileName) -* Записывает информацию в файл *fileName* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method***,** указывается, как именно следует записывать информацию

**public string UploadString(string** *address,* **string** *data) -* Записывает строку *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public string UploadString(Uri** *address,* **string** *data) -* Записывает строку *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public string UploadString(string** *address,* **string** *method,***string** *data) -* Записывает строку *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method***,** указывается, как именно следует записывать информацию

**public string UploadString(Uri** *address,* **string** *method,***string** *data) -* Записывает строку *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method***,** указывается, как именно следует записывать информацию

**public byte[] UploadValues(string** *address,* **NameValueCollection** *data) -* Записывает значения из коллекции *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public byte[] UploadValues(Uri** *address,* **NameValueCollection** *data) -* Записывает значения из коллекции *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ

**public byte[] UploadValues(string** *address,***string** *method,* **NameValueCollection** *data) -* Записывает значения из коллекции *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method*, указывается, как именно следует записывать информацию

**public byte[] UploadValues(Uri** *address,***string** *method,* **NameValueCollection** *data) -* Записывает значения из коллекции *data* по адресу URI, обозначаемому параметром *address.* В итоге возвращается ответ. В строке, передаваемой в качестве параметра *method*, указывается, как именно следует записывать информацию

**29 Опишите процесс создания потоков**

Если говорить простым языком, то поток — это некая независимая последовательность инструкций для выполнения того или иного действия в программе. В одном конкретном потоке выполняется одна конкретная последовательность действий.  
Совокупность таких потоков, выполняемых в программе параллельно называется многопоточностью программы.

Следует также запомнить, что в действительности потоки выполняются всё-таки не совсем параллельно. Дело в том, что процессор физически не может обрабатывать параллельно несколько инструкций или процессов. Однако его вычислительной мощи хватает настолько, что он может выполнять все операции по небольшому фрагменту по очереди, отводя на каждый такой фрагмент по очень маленькому кусочку времени, настолько, что кажется, будто все процессы в компьютере выполняются параллельно.

Точно такая же ситуация происходит и с потоками. Если в программе имеется 3 потока, то сначала выполняется кусочек кода из одного потока, потом кусочек кода из другого, затем — из третьего, после чего процессор снова переходит к какому-либо из двух других потоков. Выбор, какой поток необходимо назначить для выполнения в данный момент остаётся за процессором. Происходит это в доли миллисекунд, поэтому происходит ощущение параллельной работы потоков.

Основной функционал для использования потоков в приложении сосредоточен в пространстве имен System.Threading. В нем определен класс, представляющий отдельный поток - класс Thread.

Класс Thread определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем. Основные свойства класса:

* ExecutionContext: позволяет получить контекст, в котором выполняется поток
* IsAlive: указывает, работает ли поток в текущий момент
* IsBackground: указывает, является ли поток фоновым
* Name: содержит имя потока
* ManagedThreadId: возвращает числовой идентификатор текущего потока
* Priority: хранит приоритет потока - значение перечисления ThreadPriority:
  + Lowest
  + BelowNormal
  + Normal
  + AboveNormal
  + Highest

По умолчанию потоку задается значение Normal. Однако мы можем изменить приоритет в процессе работы программы. Например, повысить важность потока, установив приоритет Highest. Среда CLR будет считывать и анализировать значения приоритета и на их основании выделять данному потоку то или иное количество времени.

* ThreadState возвращает состояние потока - одно из значений перечисления ThreadState:
  + Aborted: поток остановлен, но пока еще окончательно не завершен
  + AbortRequested: для потока вызван метод Abort, но остановка потока еще не произошла
  + Background: поток выполняется в фоновом режиме
  + Running: поток запущен и работает (не приостановлен)
  + Stopped: поток завершен
  + StopRequested: поток получил запрос на остановку
  + Suspended: поток приостановлен
  + SuspendRequested: поток получил запрос на приостановку
  + Unstarted: поток еще не был запущен
  + WaitSleepJoin: поток заблокирован в результате действия методов Sleep или Join

В процессе работы потока его статус многократно может измениться под действием методов. Так, в самом начале еще до применения метода Start его статус имеет значение Unstarted. Запустив поток, мы изменим его статус на Running. Вызвав метод Sleep, статус изменится на WaitSleepJoin.

Кроме того статическое свойство CurrentThread класса Thread позволяет получить текущий поток

В программе на C# есть как минимум один поток - главный поток, в котором выполняется метод Main.

Также класс Thread определяет ряд методов для управления потоком. Основные из них:

* Статический метод GetDomain возвращает ссылку на домен приложения
* Статический метод GetDomainID возвращает id домена приложения, в котором выполняется текущий поток
* Статический метод Sleep останавливает поток на определенное количество миллисекунд
* Метод Interrupt прерывает поток, который находится в состоянии WaitSleepJoin
* Метод Join блокирует выполнение вызвавшего его потока до тех пор, пока не завершится поток, для которого был вызван данный метод
* Метод Start запускает поток

**61 Создайте приложение для изменения основных свойств компонента Bevel. Изменения должны отображаться**

**69 Создайте приложение для открытия, редактирования и сохранения текстового файла. Используйте компонент RichTextBox**

**Список использованных источников**

1. Албахари, Д. C# 5.0. Справочник. Полное описание языка / Д.Албахари. – М.: Вильямс, 2014.
2. Лабор, В.В. Си Шарп: Создание приложений для Windows / В.В.Лабор. - Мн.: Харвест, 2003.
3. Павловская, Т.А С#. Программирование на языке высокого уровня / Т.А.Павловская. - СПб: Питер, 2014.
4. Фролов, А.В. Визуальное проектирование приложений С# / А.В.Фролов. - М: КУДИЦ - ОБРАЗ, 2003.
5. Фленов, М. Библия С# / М.Фленов. - СПб.: Питер, 2011.